GPU Scheduling and the cons_tres plugin

Chad Vizino and Morris Jette
SchedMD

SLUG 2019

Thanks to NVIDIA for sponsoring this work

Goals

- More flexible scheduling mechanism
  - Especially for clusters with significant GPU resources
  - Make all CPU options available for GPUs
- Greater control over GPU resources
  - Task binding
  - Frequency control
- Improved performance of scheduling logic

Previous Shortcomings (part 1 of 4)

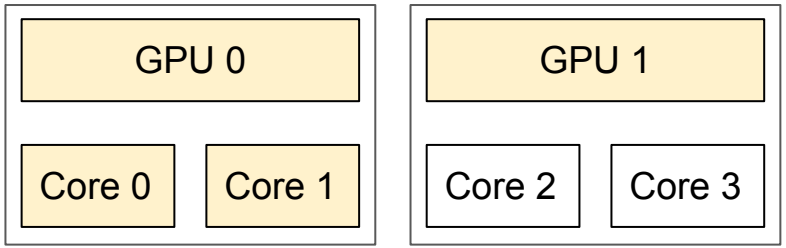
- Needed to specify GPU configuration in gres.conf file
  - Device files
  - Adjacent CPUs/cores
  - Device type
- Failed to use information available from system
  - Didn’t use NVML

Previous Shortcomings (part 2 of 4)

- GPUs allocated to jobs as a fixed count per allocated node
  - 2 GPUs per node OK
  - 4 GPUs on one node and 2 GPUs on another node not possible
  - Requesting 6 GPUs over arbitrary node count not possible
- No controls over GPU frequency or per-task binding
- No consideration of GPUs with NVLink (high speed communications between GPUs and GPUs/CPUs) to select preferred GPUs for co-scheduling

Previous Shortcomings (part 3 of 4)

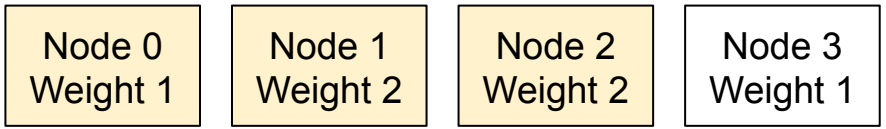
- First CPUs with adjacent GPUs are identified
- Next specific CPUs are selected for the job allocation
- Finally GPUs are selected for the job allocation
  - Favors use of CPUs and GPUs that are on same socket
  - May not be possible since GPUs are selected after CPUs



Cores 0 and 1 might be selected for a job allocation requiring 2 GPUs rather than selecting one core on each socket with a GPU

Previous Shortcomings (part 4 of 4)

- Node scheduling weights used in sub-optimal fashion
  - If job allocation can't be satisfied with lowest weight nodes then all nodes with the lowest plus next lowest weight considered on equivalent basis for use without allocating as many of the lowest weight nodes first



A 3-node job allocation might not use all of the lowest weight nodes

New select/cons_tres Plugin

- “cons_tres” represents “Consumable TRES”
- “TRES” represents “Trackable RESources”
- All functionality provided by “cons_res” plugin is also supported by “cons_tres” (e.g. CR_LLN, CR_PACK_NODES, CR_SOCKET, etc.)
- Addresses all of the previously cited shortcomings
- New “gpu” job options only supported by the cons_tres plugin
  - No other select plugin recognizes the new GPU options

New Job Submit Options

Same options apply to salloc, sbatch and srun commands

- --cpus-per-gpu= CPUs required per allocated GPU
- -G/--gpus= GPU count across entire job allocation
- --gpu-bind= Task/GPU binding option
- --gpu-freq= Specify GPU and memory frequency
- --gpus-per-node= Works like “--gres=gpu:#” option today
- --gpus-per-socket= GPUs per allocated socket
- --gpus-per-task= GPUs per spawned task
- --mem-per-gpu= Memory per allocated GPU

New Configuration Parameters

Parameters available globally and on per-partition basis. Command line options override these default values.

- DefCpusPerGPU= Default CPUs count per allocated GPU
- DefMemPerGPU= Default memory size per allocated GPU

Data Structure Changes (part 1 of 3)

The new “GPU” options are translated by job submit commands to “tres” options and reported by “scontrol show job”, sview, and squeue accordingly.

- *--gpus-per-node=4* translated to *tres-per-node=gpu:4*
- *--mem-per-gpu=12* translated to *mem-per-tres=gpu:12*

Data Structure Changes (part 2 of 3)

- cons_tres plugin tracks resources on a per-socket (rather than per-node) basis and accumulates co-located GPUs and cores when possible
- Resource availability currently managed as list of GRES, but could easily be modified to track other resources (e.g. memory, licenses, etc.)

Data Structure Changes (part 3 of 3)

- Cons_tres uses per-node array of core-bitmaps to track resources rather than a single cluster-wide core-bitmap
  - Future: Will support changing core count on a node without restarting slurmctld daemon (important for cloud)

Examples of Use (part 1 of 2)

```
$ sbatch --ntasks=16 --gpus-per-task=2 my.bash

$ sbatch --ntasks=8 --ntasks-per-socket=2 --gpus-per-socket=tesla:4 my.bash

$ sbatch --gpus=16 --gpu-freq=low,verbose --gpu-bind=closest --nodes=2 my.bash

$ sbatch --gpus=gtx1080:8,gtx1060:2 --nodes=1 my.bash
```

Examples of Use (part 2 of 2)

Allocation of resources to job steps also supports these GPU options:

```
$ cat my.bash
#!/bin/bash
srun --gpus=1 --ntasks=1 --nnodes=1 app1 &
srun --gpus=1 --ntasks=1 --nnodes=1 app2 &
srun --gpus=2 --ntasks=1 --nnodes=1 app3 &
srun --gpus=2 --ntasks=1 --nnodes=1 app4 &
wait

$ sbatch --gpus=2 my.bash
```

Conflicting Options (part 1 of 2)

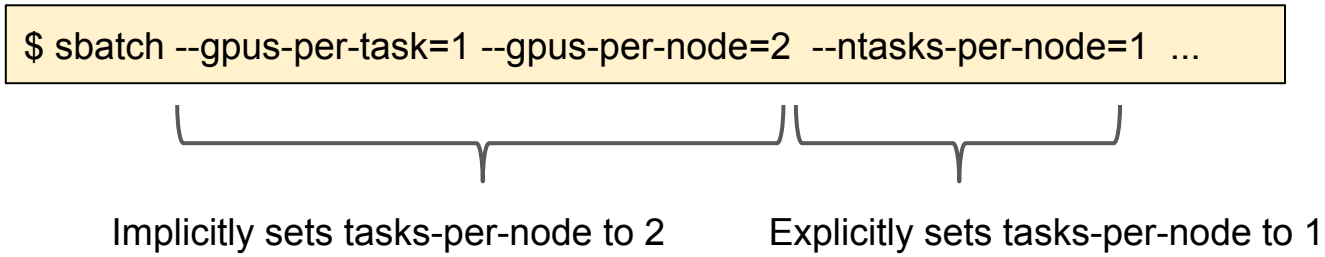
- Given the multitude of options, it is possible to submit a job with conflicting options
  - Many conflicting options are possible
  - In most cases the job will be rejected

```
$ sbatch --gpus-per-task=1 --cpus-per-gpu=2  --cpus-per-task=1  ...
```

Implicitly sets cpus-per-task to 2

Explicitly sets cpus-per-task to 1

Conflicting Options (part 2 of 2)



CUDA MPS Support - Overview

- Multi-Process Service (MPS) is an NVIDIA feature that supports simultaneously running multiple CUDA programs on a shared GPU
- Each job can be allocated some percentage of GPU threads
- Binary compatible with CUDA API
- Requires GPU with compute capability version 3.5 or higher

NVIDIA's MPS documentation:
https://docs.nvidia.com/deploy/pdf/CUDA_Multi_Process_Service_Overview.pdf

Configuration Example (1 of 2)

```
# Excerpt from slurm.conf
GresTypes=gpu,mps
NodeName=nid[00-64] Gres=gpu:2,mps:2200
```

```
# Excerpt from gres.conf
# Explicitly specify different counts on each GPU in MPS mode
Name=gpu File=/dev/nvidia0 Type=p40   Cores=0-3
Name=gpu File=/dev/nvidia1 Type=p100 Cores=4-7
Name=mps File=/dev/nvidia0 Count=1000  # Type and Cores copied from GPU above
Name=mps File=/dev/nvidia1 Count=1200  # Type and Cores copied from GPU above
```

Configuration Example (2 of 2)

```
# Excerpt from slurm.conf
GresTypes=gpu,mps
NodeName=nid[00-64] Gres=gpu:2,mps:200
```

```
# Excerpt from gres.conf
AutoDetect=nvml
# Automatically detects GPUs, their type, cores, and NVLinks
# MPS resources evenly distributed over the 2 GPUs found
```

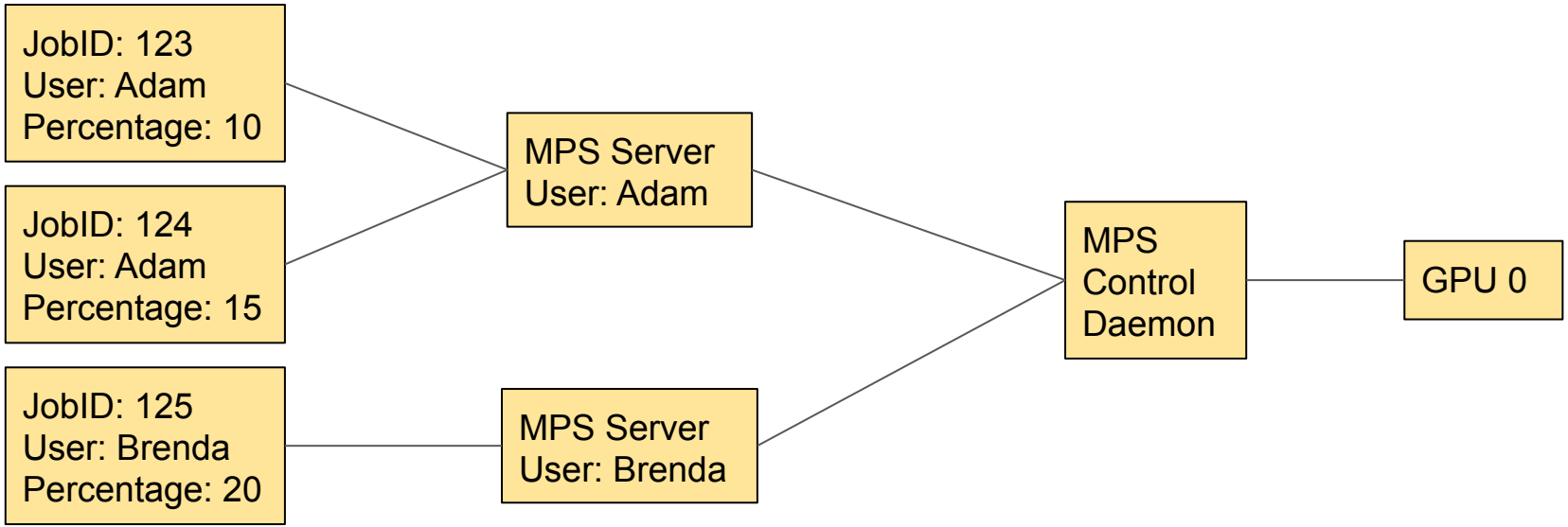
Usage

- Specify MPS requirements like any other GRES
  - Name “mps” and count are required
  - Type (Model) information is optional

```
$ sbatch --gres=mps:40 …

$ sbatch --gres=mps:p100:20
```

MPS Example



Summary

- More flexible scheduling mechanism
  - Especially for clusters with GPU resources
- Greater control over GPU resources
  - New GPU options for jobs
  - Integration with MPS
  - Use NVML to determine GPU resources
- Improved performance of scheduling logic
  - New data structures for more flexibility and speed

Questions?